Node Debug 三法三例

wizardforcel

Published
with GitBook

目錄

介绍	0
node debug	1
node inspector	2
测试驱动开发	3
资源	4

1. Node Debug 三法三例

> 作者：i5ting
>
> 来源：node-debug tutorial

大家对nodejs调试应该都比较头疼，至少我这个不用IDE写js的人很头疼这个，其实node的生态圈非常好 有非常好的工具和非常潮的开发方式

这里总结了3法3例，希望能对大家有所帮助

3种方法

- node debugger
- node inspector
- 测试驱动开发

3个例子

- hello world
- 继承例子
- express helloworld

1.1. node debug

V8 提供了一个强大的调试器，可以通过 TCP 协议从外部访问。Nodejs提供了一个内建调试器来帮助开发者调试应用程序。想要开启调试器我们需要在代码中加入debugger标签，当Nodejs执行到debugger标签时会自动暂停（debugger标签相当于在代码中开启一个断点）。

1.1.1. hello world例子

代码如下：

see `helloword-debug.js`

```
var hello = 'hello';
var world = 'nodejs';

debugger;

var hello_world = hello + ' ' + world;
console.log(hello_world);
```

执行命令：`node debug helloword-debug.js` 就可以进入调试模式。

当然，首先需要在程序代码中手动添加中断debugger; ， 这样当以调试模式运行时，程序会自动中断，然后等候你调试，就像GDB一样，可以用help命令查看自己都可以使用哪些调试命令。

```
node-debug-tutorial git:(master) ? node debug helloword-debug.js
< debugger listening on port 5858
connecting... ok
break in helloword-debug.js:1
  1 var hello = 'hello';
  2 var world = 'nodejs';
  3
debug> help
Commands: run (r), cont (c), next (n), step (s), out (o), backtrace (bt), setBreakpoint (
watch, unwatch, watchers, repl, restart, kill, list, scripts, breakOnException, breakpoin
debug>
debug> n
break in helloword-debug.js:2
  1 var hello = 'hello';
  2 var world = 'nodejs';
  3
  4 debugger;
debug> repl
Press Ctrl + C to leave debug repl
> hello
'hello'
```

此时repl打开js上下文即时求值环境，和chrome的debug的console是一样的。

如果想退出，请按下 `ctrl + c` ,这样就可以返 到debug模式

```
debug> n
break in helloword-debug.js:4
  2 var world = 'nodejs';
  3
  4 debugger;
  5
  6 var hello_world = hello + ' ' + world;
debug> n
break in helloword-debug.js:6
  4 debugger;
  5
  6 var hello_world = hello + ' ' + world;
  7 console.log(hello_world);
  8
debug> n
break in helloword-debug.js:7
  5
  6 var hello_world = hello + ' ' + world;
  7 console.log(hello_world);
  8
  9 });
debug> repl
Press Ctrl + C to leave debug repl
> hello_world
'hello nodejs'
>
end
```

如果想终止调试，请按下2次 `ctrl + c` 键

1.1.2. 命令说明

可选项	用途
run	执行脚本,在第一行暂停
restart	重新执行脚本
cont, c	继续执行,直到遇到下一个断点
next, n	单步执行
step, s	单步执行并进入函数
out, o	从函数中步出
setBreakpoint(), sb()	当前行设置断点
setBreakpoint('f()'), sb(...)	在函数f的第一行设置断点
setBreakpoint('script.js', 20), sb(...)	在 script.js 的第20行设置断点
clearBreakpoint, cb(...)	清除所有断点
backtrace, bt	显示当前的调用栈
list(5)	显示当前执行到的前后5行代码
watch(expr)	把表达式 expr 加入监视列表
unwatch(expr)	把表达式 expr 从监视列表移除
watchers	显示监视列表中所有的表达式和值
repl	在当前上下文打开即时求值环境
kill	终止当前执行的脚本
scripts	显示当前已加载的所有脚本
version	显示v8版本

这里就和gdb等调试器一模一样了

回归一下，debug的2种模式：

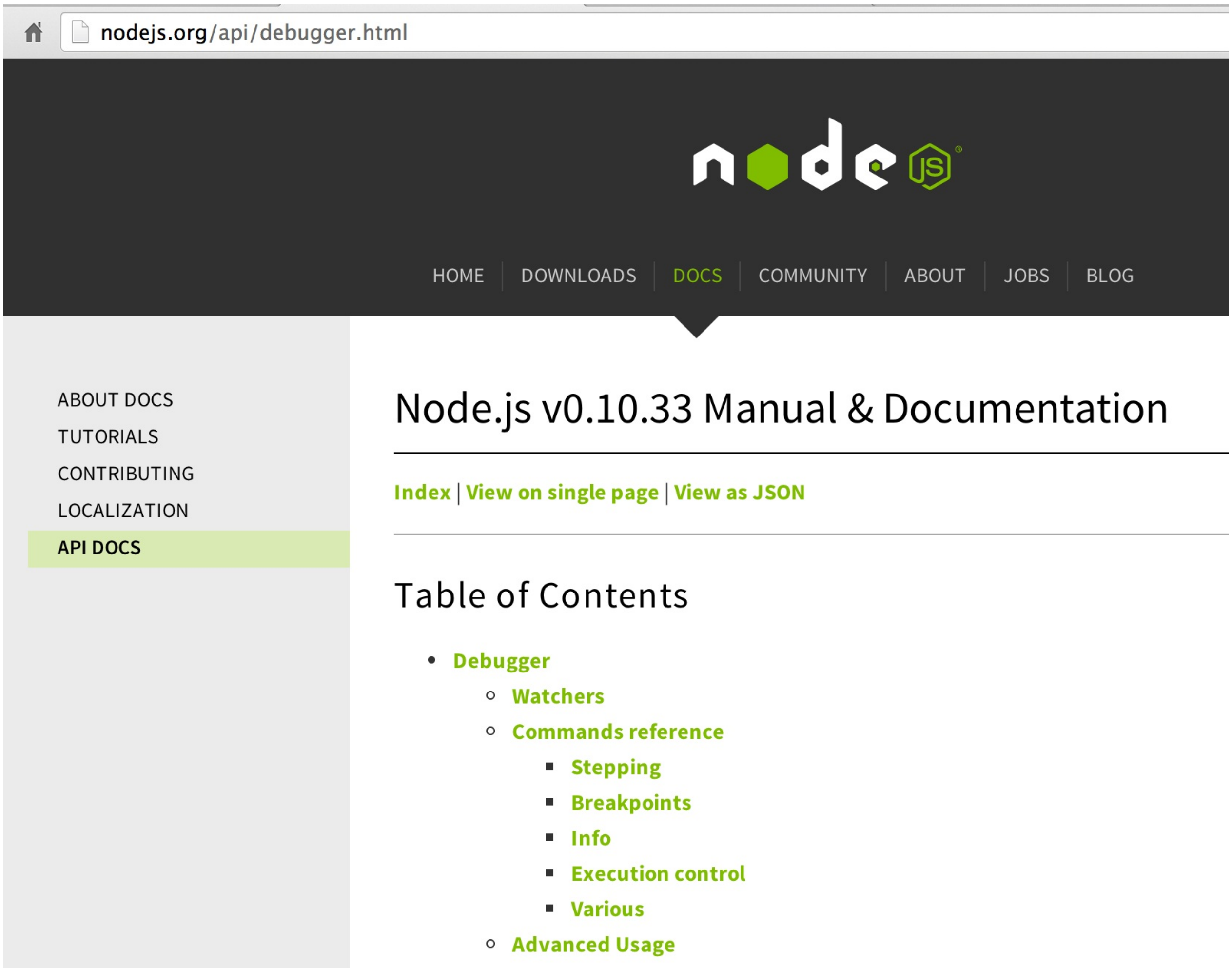
- js上下文即时求值环境模式
- debug断点模式

八卦一下啊，你了解vi的3种工作模式么？

- 普通(normal)模式，又称命令模式
- 插入(insert)模式
- 命令行(cmdline)模式

化用一下更容易理解

node debugger官方文档



上面有更多的例子和api，有了上面的基础，学习会非常简单。

1.1.3. FAQ

注意,如果出现

```
< Failed to open socket on port 5858, waiting 1000 ms before retrying
```

请结束掉所有debug进程

```
ps -ef|grep debug-brk|awk '{print $2}'|xargs kill -9
```

1.2. node inspector

上面这种方式稍微有些麻烦，作为前端开发人员，我们写JS代码调试的时候一般都用FireBug或Chrome浏览器内置的调试工具，其实nodejs程序也可以这样子来调试，但是首先需要安装一个node-inspector的模块。

node-inspector是通过websocket方式来转向debug输入输出的。因此，我们在调试前要先启动node-inspector来监听Nodejs的debug调试端口。

1.2.1. 安装

这个需要用npm来安装，只需要执行下列语句：

```
npm install -g node-inspector
```

安装完成之后，通常可以直接这样启动在后台：

```
node-inspector &
```

默认会监听8080端口，当然，也可能通过使用--web-port参数来修改。然后，在执行node程序的时候，多加个参数：--debug-brk, 如下：

```
node --debug-brk app.js
```

或者

```
node-debug app.js
```

控制台会返回“debugger listening on port 5858”， 现在打开浏览器，访问 http://localhost:8080/debug?port=5858，这时候就会打开一个很像Chrome内置调试工具的界面，并且代码断点在第一行，下面就可以使用这个来调试了。

常用调试功能

- 增加断点，查看调用栈，变量等
- 使用console打印查看日志

使用方法和 `chrome` 的 `inspect element` 调试web开发是一样的。

调试还是很方便的，而且可以远程调试。其实原理很简单，它启动的是一个web server，我们要做的就是把localhost换成对应ip即可，要注意服务器的防火墙哦。

1.2.2. 测试extend.js

测试一下继承是否ok，首先执行命令，打印出结果，但这种办法才挫了

```
?  node-debug-tutorial git:(master) node extend.js
node debug
hello node debug
```

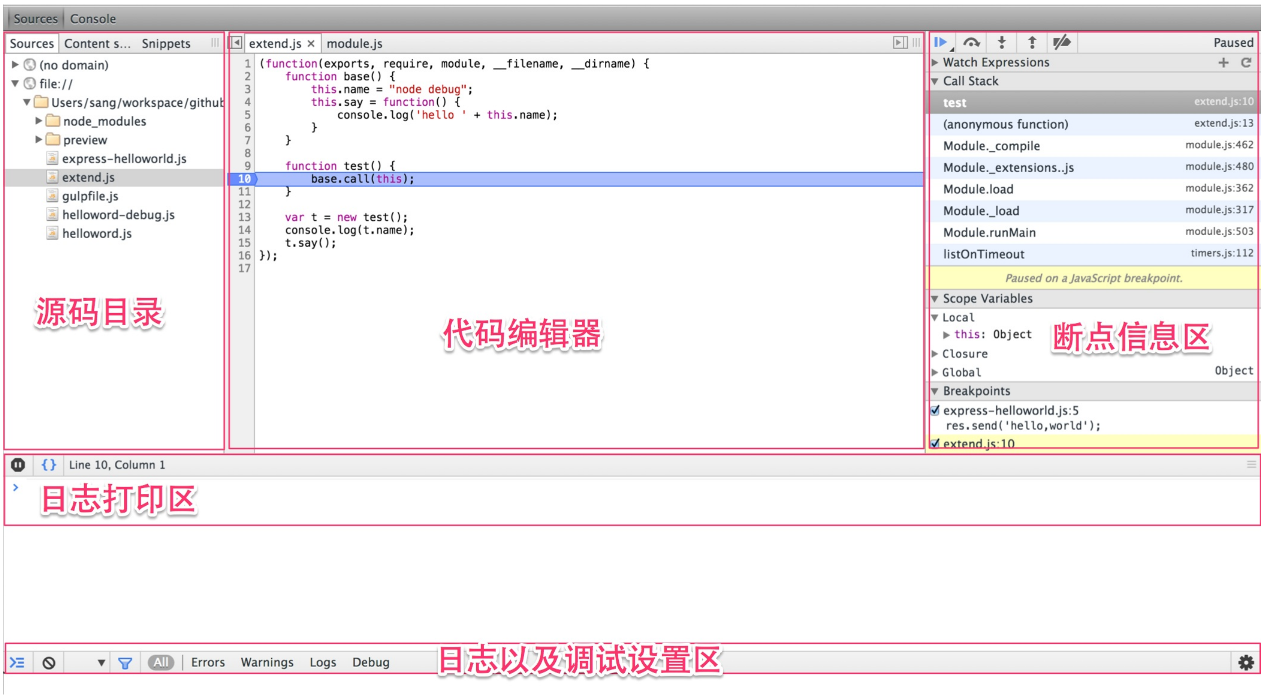
开始使用node-inspector调试

1.2.2.1. 启动

```
?  node-debug-tutorial git:(master) node-debug  extend.js
Node Inspector is now available from http://localhost:8080/debug?port=5858
Debugging `extend.js`

debugger listening on port 5858
```

1.2.2.2. 界面说明



mac系统大部分人都记不住这些按键，下面给出说明

Symbol	Key
?	Command Key
?	Control Key
?	Option Key
?	Shift Key

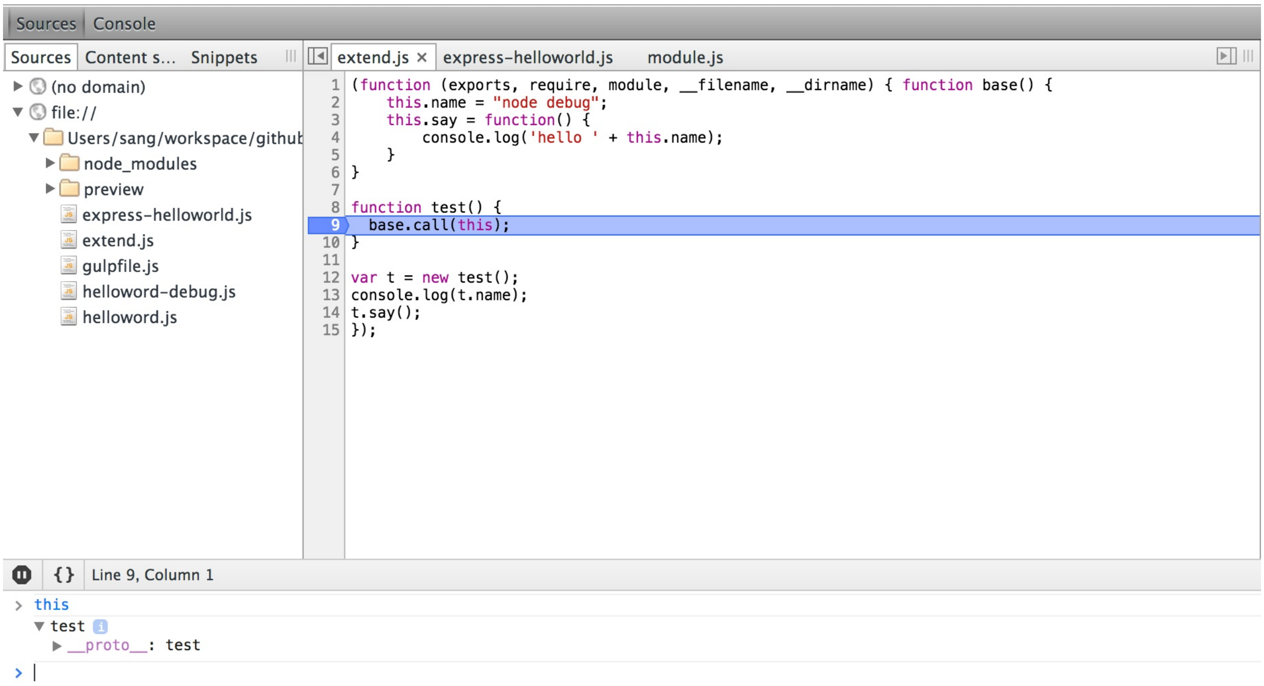
断点操作

- resume script execution（F8） 挂起断点，也可以理解为放弃当前断点，如果有下一个断点，会自动断住得
- step over（F10） 跳过这行，到下一行，如果当前函数结束，会跳到调用栈的上一级的下一行
- step into（F11） 进入当前行代码里的函数内部
- step out（Shift + F11） 从当前函数退出到之前进入的代码处

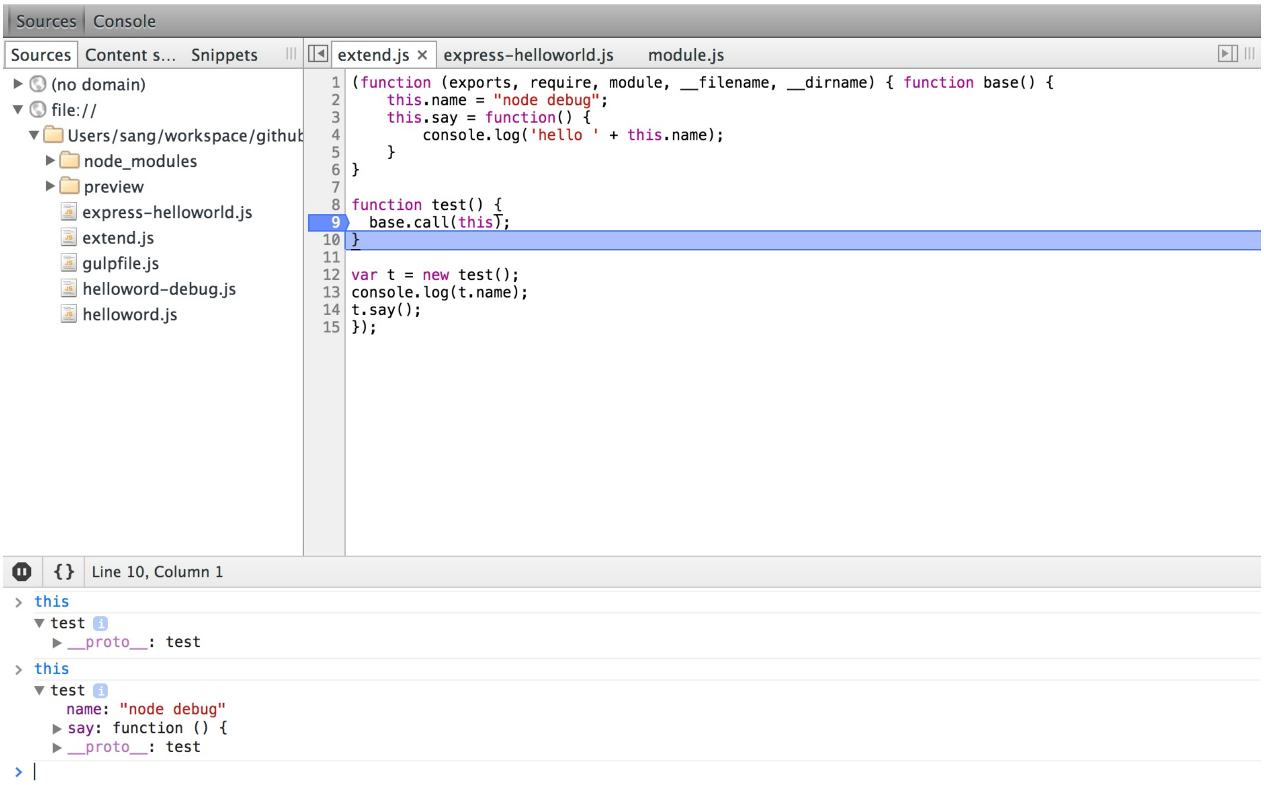
控制台操作

- 不能使用var，直接打印变量杰克

1.2.2.3. 增加断点，并打印出this



1.2.2.4. 断点下一步，并打印出this



1.2.2.5. 结论

通过

```
base.call(this);
```

这行代码，明显看到test对象的this被改变了，即使test拥有了base的所有属性和方法，这是最简单的实现继承的方法，当然多重继承mixin也可以是这样的原理

1.2.3. 测试express helloworld

这种测试一般都是看request里的params，query和body等

准备工作

```
npm init .
npm install --save express
touch express-helloworld.js
```

测试express-helloworld.js代码

```
var express = require('express');
var app = express();

app.get('/',function(req,res){
    res.send('hello,world');
});

app.listen(5008);
```

执行,安装服务器自动重载模块

```
npm install -g supervisor
supervisor express-helloworld.js
```

打开浏览器访问http://127.0.0.1:5008/就会看到helloworld返回

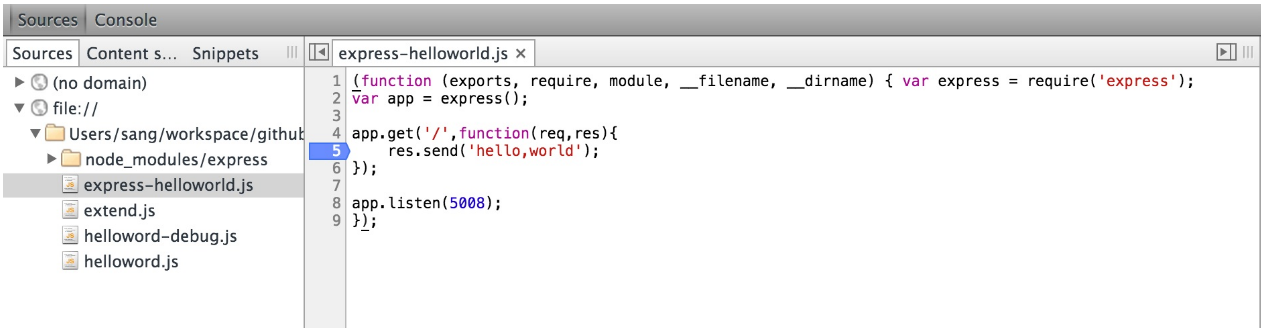
此时终止 `supervisor express-helloworld.js` ,使用 `ctrl + c` 终止。

然后使用node-inspect调试

```
?  node-debug-tutorial git:(master) ? node-debug express-helloworld.js
Node Inspector is now available from http://localhost:8080/debug?port=5858
Debugging `express-helloworld.js`

debugger listening on port 5858
```

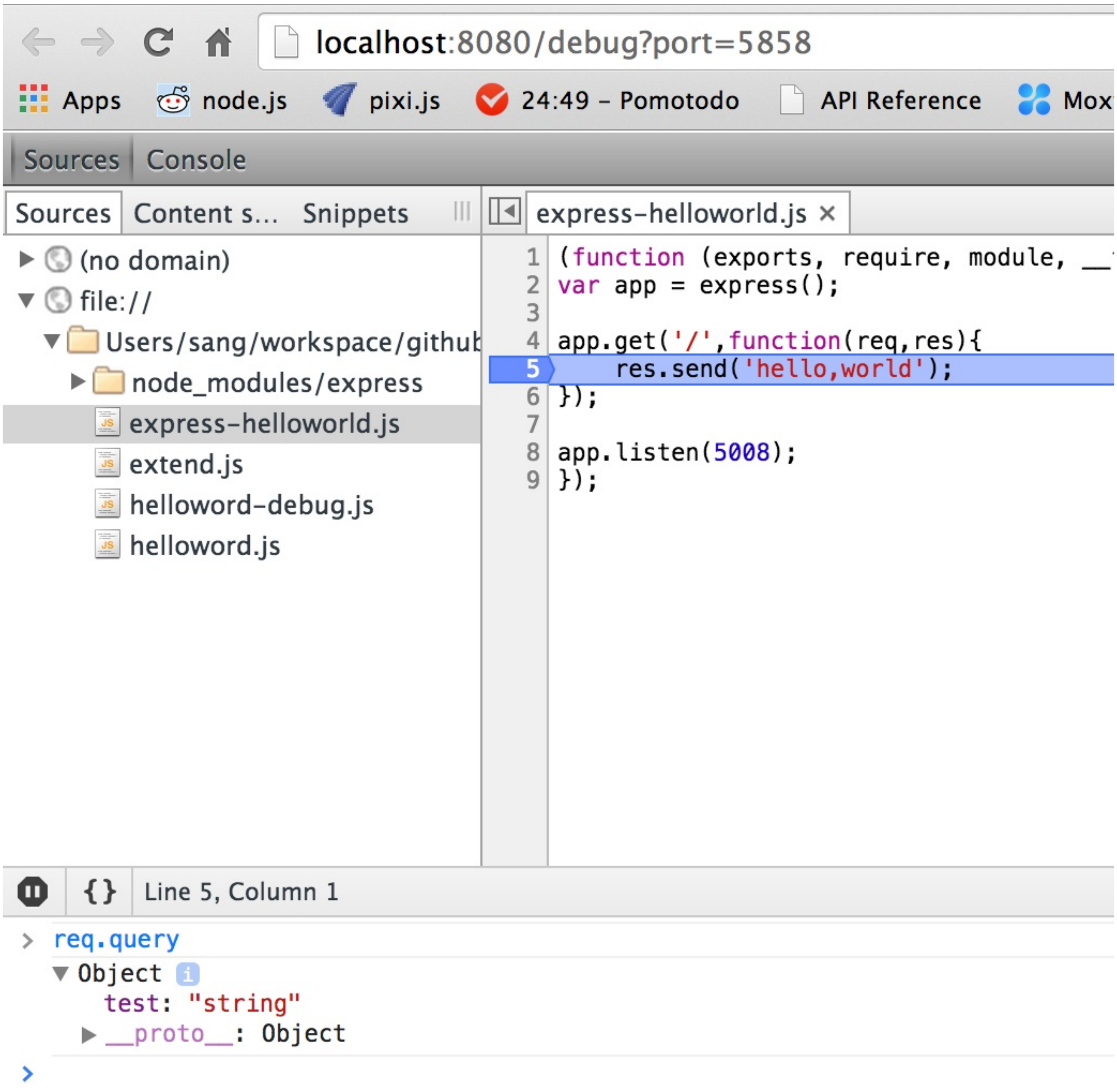
增加断点



使用curl来模拟get请求，增加一个参数test，便于一会的debug

```
curl -G -d "test=string" http://127.0.0.1:5008/
```

此时浏览器页面会停在断点处，在console里输入 `req.query` 即可以查到参数


localhost:8080/debug?port=5858
Apps
node.js
pixi.js
24:49 – Pomotodo
API Reference
Sources
Console
Sources
Content s…
Snippets
express-helloworld.js
(no domain)
file://
Users/sang/workspace/githuk
node_modules/express
express-helloworld.js
extend.js
helloword-debug.js
helloword.js
(function (exports, require, module, __
var app = express();
app.get('/',function(req,res){
res.send('hello,world');
});
app.listen(5008);
});
Line 5, Column 1
req.query
Object
test: "string"
__proto__: Object

1.3. 测试驱动开发

- tdd
- bdd
- 代码覆盖率

1.3.1. 测试框架

- nodeunit
- mocha

1.3.2. 更多测试

```
npm install --save-dev chai
npm install --save-dev sinon
npm install --save-dev supertest
npm install --save-dev zombie
```

1.3.3. 代码覆盖率

修改Gulpfile.js

- auto test
- 代码测试覆盖率

```
npm install --save-dev gulp
npm install --save-dev gulp-mocha
npm install --save-dev gulp-istanbul
```

创建gulpfilejs

```
var gulp = require('gulp');
var istanbul = require('gulp-istanbul');
var mocha = require('gulp-mocha');

gulp.task('test', function (cb) {
  gulp.src(['db/**/*.js'])
    .pipe(istanbul()) // Covering files
    .on('finish', function () {
      gulp.src(['test/*.js'])
        .pipe(mocha())
        .pipe(istanbul.writeReports()) // Creating the reports after tests runned
        .on('end', cb);
    });
});

gulp.task('default',['test'], function() {
  gulp.watch(['./db/**/*','./test/**/*'], ['test']);
});

gulp.task('watch',['test'], function() {
  gulp.watch(['./db/**/*','./test/**/*'], ['test']);
});
```

测试

node_modules/.bin/gulp 这时，你试试修改测试或源文件试试，看看会不会自动触发测试

当然，如果你喜欢只是测试一次，可以这样做

node_modules/.bin/gulp test 如果你不熟悉gulp，可以再这里[https://github.com/i5ting/js-tools-best-practice/blob/master/doc/Gulp.md学习](https://github.com/i5ting/js-tools-best-practice/blob/master/doc/Gulp.md)

修改package.json

```
"scripts": { "start": "./node_modules/.bin/supervisor ./bin/www", "test": "./node_modules
```

1.4. 资源

- debugger官方文档
- node-inspector仓库地址
- nodeunit
- mocha
- https://github.com/baryon/tracer
- http://www.habdas.org/node-js-debugging-primer/
- https://github.com/visionmedia/mocha
- http://visionmedia.github.io/mocha/
- http://mochajs.org/
- https://github.com/chaijs/chai
- http://chaijs.com/
- http://sinonjs.org/
- http://zombie.labnotes.org/
- https://github.com/tj/supertest（api test文档）
- https://github.com/tj/superagent/blob/master/test/node/agency.js（api test示例）
- https://github.com/i5ting/js-tools-best-practice/blob/master/doc/Gulp.md
- https://github.com/SBoudrias/gulp-istanbul